

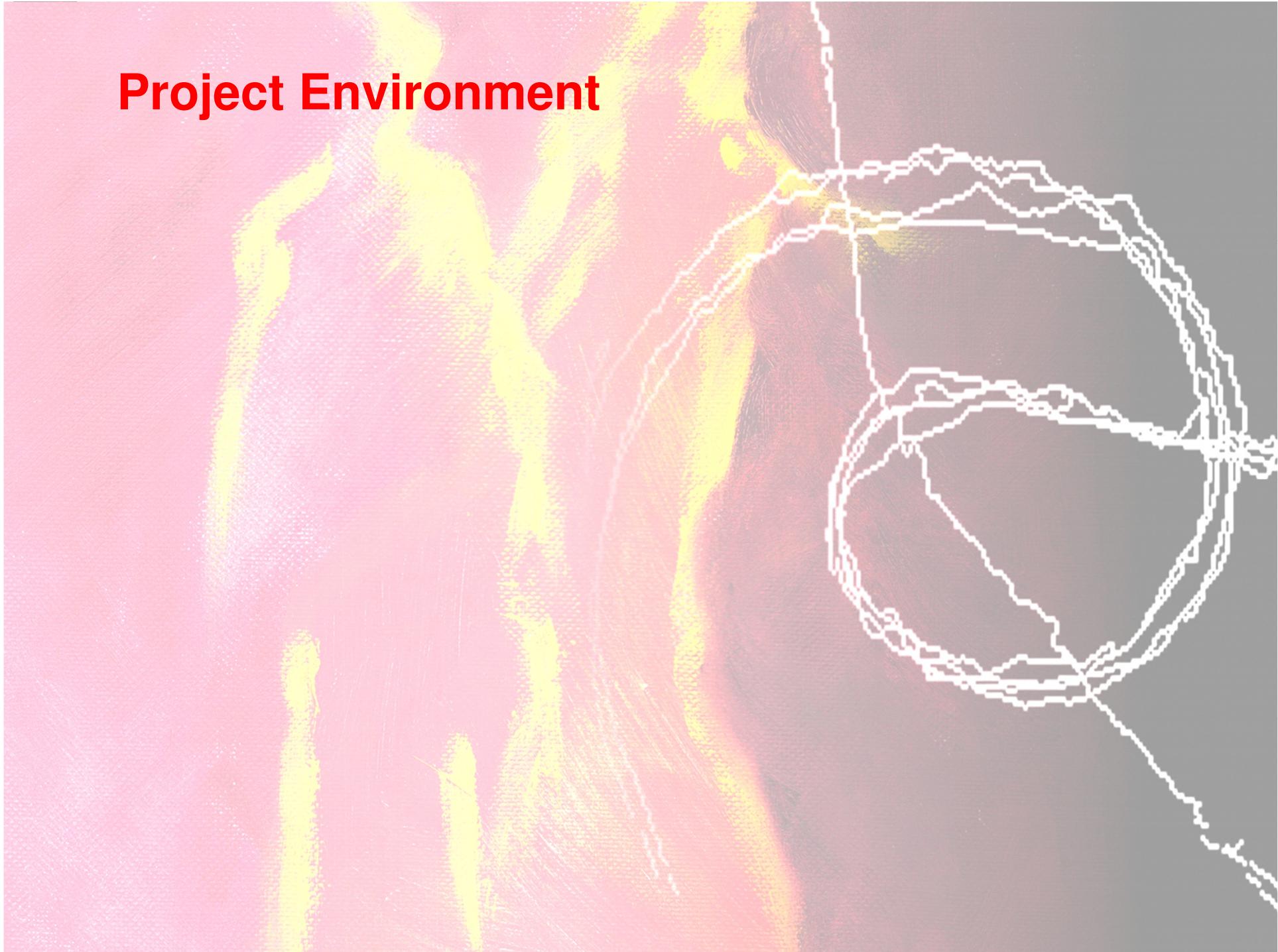
COSMO Dynamical Core Redesign

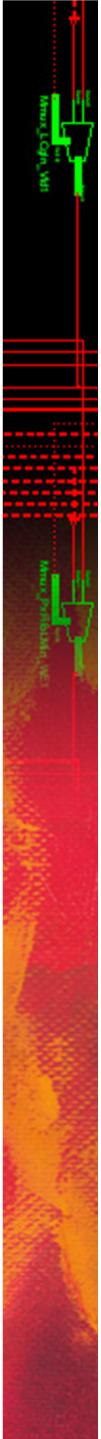
Tobias Gysi
David Müller
Boulder, 8.9.2011

Supercomputing Systems AG
Technopark 1
8005 Zürich

Fon +41 43 456 16 00
Fax +41 43 456 16 10
www.scs.ch

Project Environment





Who is SCS?

Company Profile

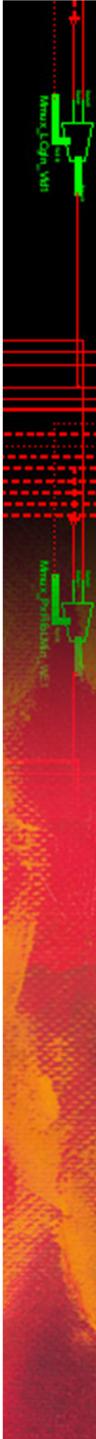
- Privately owned R&D services company founded in 1993.
- More than 70 engineers and scientists in Technopark Zurich.
- 100% dedicated to our customer projects.
- All IP-Rights (including source-code, documentation, etc.) belong to our customers.

Approach

- Innovation processes
- System design, HW, FW & SW design
- Implementation & test
- Lifetime support

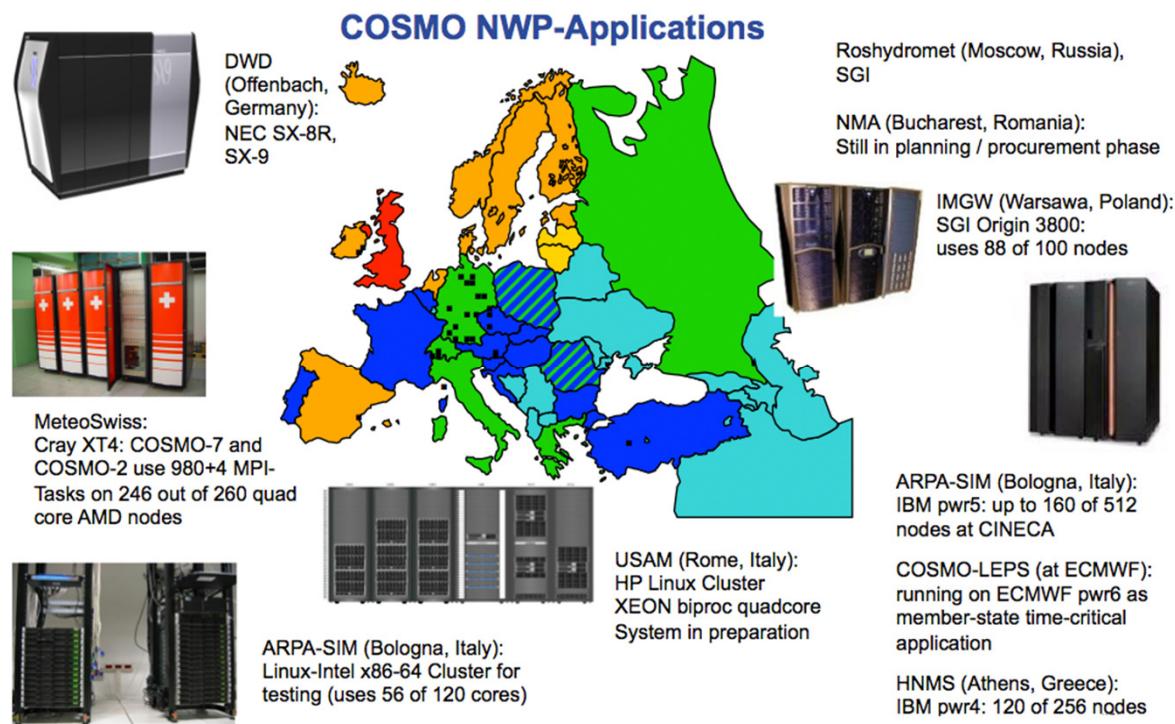
Core Competencies

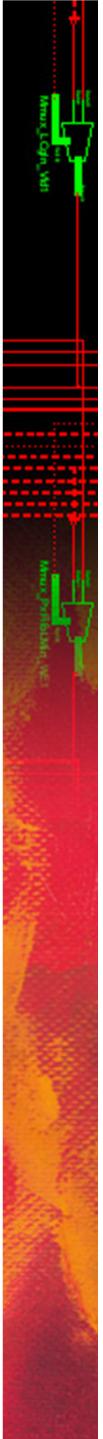
- Fast communication & data processing
- Enterprise applications
- Scaling & highly reliable system architectures
- Algorithm optimizations



The COSMO model

- Limited-area model - <http://www.cosmo-model.org/>
- Operational at 7 national weather services
- O(50) universities and research institutes

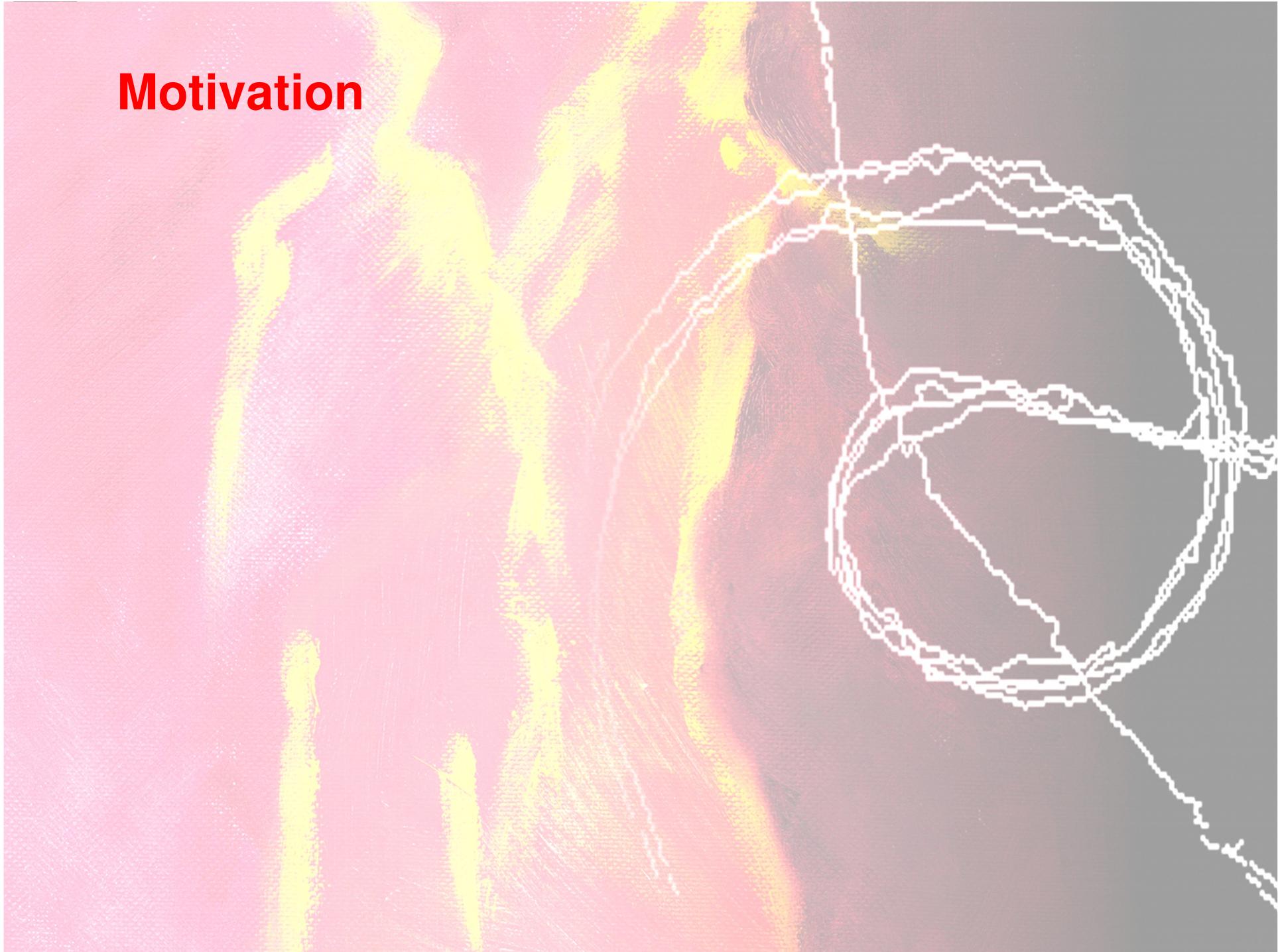


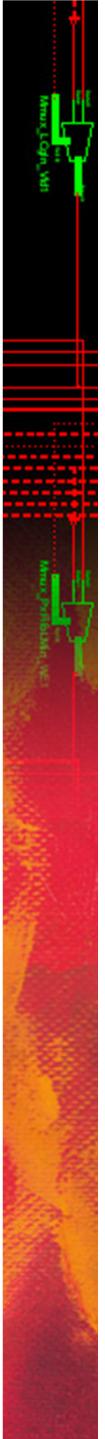


HP2C

- 10 Projects from different domains - <http://www.hp2c.ch/>
 - Cardiovascular simulation (EPFL)
 - Stellar explosions (University of Basel)
 - Quantum dynamics (University of Geneva)
 - ...
 - COSMO-CCLM
- COSMO-CCLM tasks
 1. Cloud resolving climate simulations (IPCC AR5)
 2. Adapt existing code (hybrid, I/O)
 3. Rewrite of dynamical core

Motivation





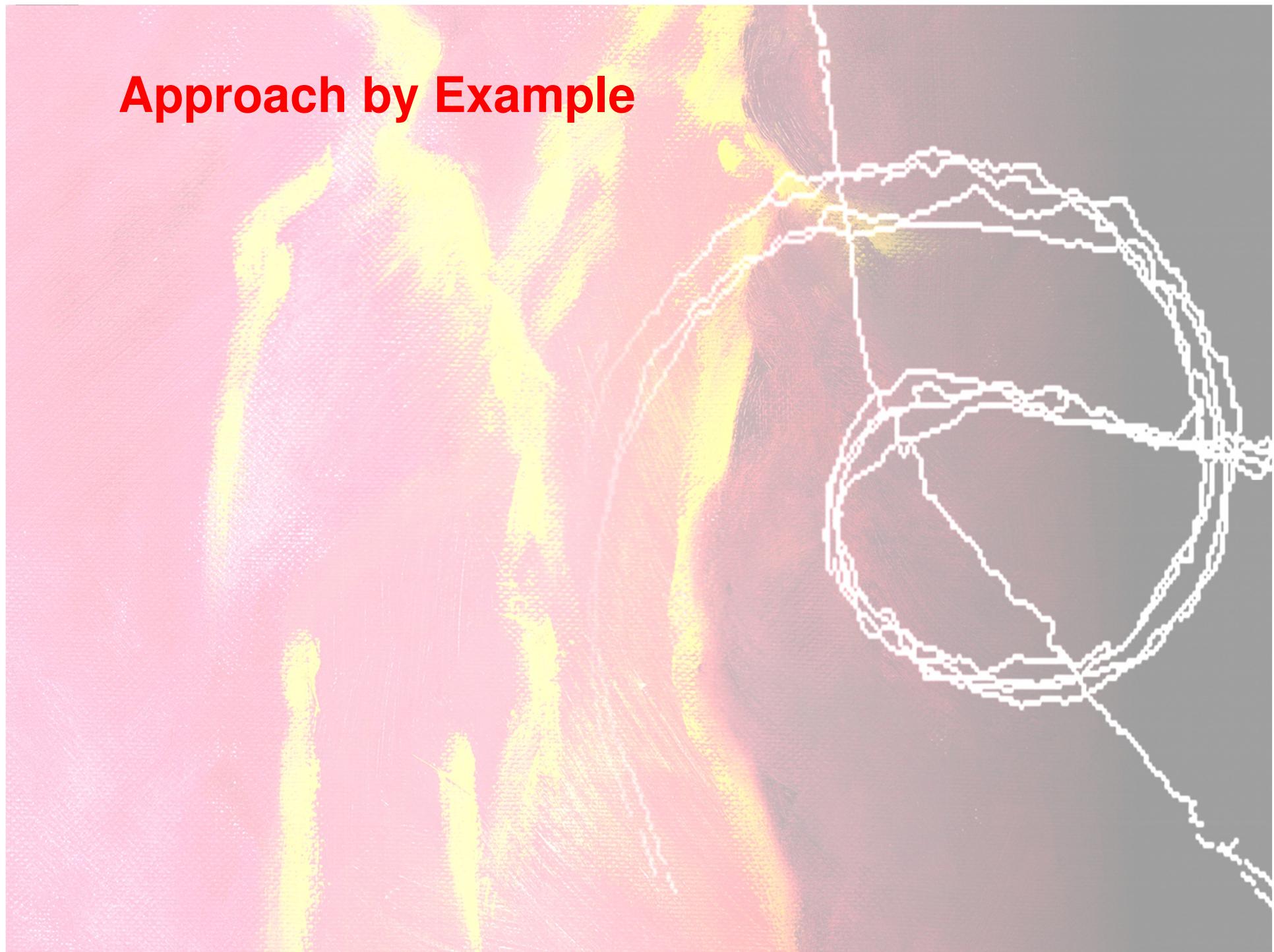
Feasibility Study

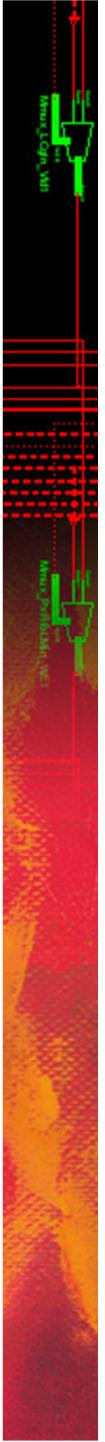
- The code is memory bandwidth limited
- Prototype implementation of the Fast Wave Solver showed
 - 2x performance increase through code optimization
 - Performance scales with memory bandwidth (strong interest in GPUs)
- But the optimizations are
 - Architecture dependent
 - Increase code complexity
- Conclusion
 - Try to rewrite the code while hiding the additional complexity

Goals of the HP2C Dycore rewrite

	Goal	Measures	
1	Correctness	Unit testing	
		Verification framework	
2	Performance	Loop merging	
		Calculation on-the-fly	
3	(Performance) Portability x86-GPU-...	Split user application from platform specific library	
		Allow different storage orders and blocking schemes	
		Support 3-level parallelism	Nodes Cores SIMD
4	Ease of Use	Single source code	Readability
			Usability
			Maintainability

Approach by Example





An Example – Fast Wave U-Update

- X-component (transformed into spherical, terrain-following coordinates)

$$\frac{\partial u}{\partial t} = -\frac{1}{\rho a \cos \phi} \left(\frac{\partial p'}{\partial \lambda} + \frac{\partial z}{\partial \lambda} \frac{\partial p'}{\partial \zeta} \right)$$

- X-component (discretized form)

$$\frac{\partial u}{\partial t} = -\frac{1}{\bar{\rho}^\lambda a \cos \phi} \frac{180}{\pi d\lambda} \left(\delta_\lambda p' + \frac{\delta_\lambda \bar{z}^\zeta}{\delta_\zeta \bar{z}^\lambda} \overline{\delta_\zeta p'}^\lambda \right)$$

- Fundamental operators

$$\delta_\lambda x = x_{i+1,k} - x_{i,k} \quad \bar{x}^\lambda = \frac{1}{2} (x_{i+1,k} + x_{i,k})$$

$$\delta_\zeta x = x_{i,k+1} - x_{i,k} \quad \bar{x}^\zeta = \frac{1}{2} (x_{i,k+1} + x_{i,k})$$

Fortran Version

```
DO k = MAX(2,kflat), ke
    zppgradcor(istart-1:iend+1,jstart-1:jend+1,k) = &
        pp(istart-1:iend+1,jstart-1:jend+1,k) * wgtfac(istart-1:iend+1,jstart-1:jend+1,k) +  &
        pp(istart-1:iend+1,jstart-1:jend+1,k-1) * (1.0_ireals-wgtfac(istart-1:iend+1,jstart-1:jend+1,k))
ENDDO
DO k = 1, ke
    IF ( k < kflat ) THEN
        DO j = jstartu, jendu
            DO i = ilowu, iendu
                zpgradx = (zpi(i+1,j,k) - zpi(i,j,k)) * zrhoqx_i(i,j,k) * dts
                u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx + suten(i,j,k)
            END DO
        END DO
    ELSE
        zdpdz(istart-1:iend+1,jstart-1:jend+1) = &
        zppgradcor(istart-1:iend+1,jstart-1:jend+1,k+1) - zppgradcor(istart-1:iend+1,jstart-1:jend+1,k)

        DO j = jstartu, jendu
            DO i = ilowu, iendu
                zdpdzu = zdpdz(i+1,j) + zdpdz(i,j)
                zdzpz = zdpdzu * zdzdx(i,j,k)
                zdpx = pp(i+1,j,k) - pp(i,j,k)
                zpgradx = zdpx + zdzpz
                u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx*zrhoqx_i(i,j,k) * dts + suten(i,j,k)
            END DO
        END DO
    ENDIF
ENDDO
```

2 full 3D loops

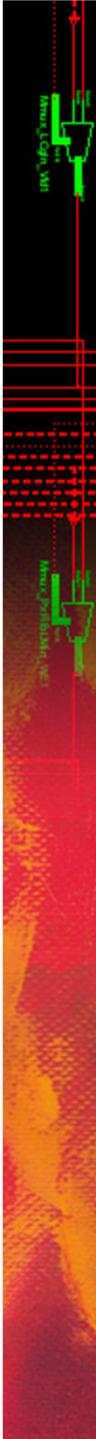
Fortran Version

Actual stencil definitions – User Application

Loops – Architecture Dependent

```
DO k = MAX(2,kflat), ke
    zppgradcor(istart-1:iend+1,jstart-1:jend+1,k) = &
        pp(istart-1:iend+1,jstart-1:jend+1,k) * wgtfac(istart-1:iend+1,jstart-1:jend+1,k) +  &
        pp(istart-1:iend+1,jstart-1:jend+1,k-1) * (1.0_ireals-wgtfac(istart-1:iend+1,jstart-1:jend+1,k))
ENDDO
DO k = 1, ke
    IF ( k < kflat ) THEN
        DO j = jstartu, jendu
            DO i = ilowu, iendu
                zpgradx = (zpi(i+1,j,k) - zpi(i,j,k)) * zrhoqx_i(i,j,k) * dts
                u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx + suten(i,j,k)
            ENDDO
        ENDDO
    ELSE
        zdpdz(istart-1:iend+1,jstart-1:jend+1) = &
        zppgradcor(istart-1:iend+1,jstart-1:jend+1,k+1) - zppgradcor(istart-1:iend+1,jstart-1:jend+1,k)

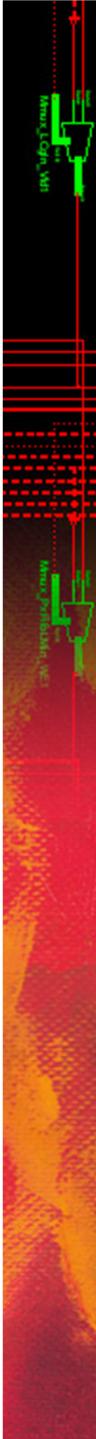
        DO j = jstartu, jendu
            DO i = ilowu, iendu
                zdpdzu = zdpdz(i+1,j) + zdpdz(i,j)
                zdzpz = zdpdzu * zdzdx(i,j,k)
                zdpx = pp(i+1,j,k) - pp(i,j,k)
                zpgradx = zdpx + zdzpz
                u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx*zrhoqx_i(i,j,k) * dts + suten(i,j,k)
            ENDDO
        ENDDO
    ENDIF
ENDDO
```



Idea

- Write a library to abstract the underlying hardware platform (CPU / GPU)
 - Adapt loop order / storage layout to the platform
 - Optimize data field access / index calculation
 - Leverage software caching
- Problem
 - There is no straight forward stencil library API
- Approach split loop logic from update functions
 - Functors wrap the update functions
 - Domain Specific Embedded Language (DSEL) specifies the loops / control flow

C++ Metaprogramming allows to define a “loop language” / DSEL which generates the loops at compile time



Stencil Definition and Usage

```
Stencil fastWaveUUpdate;
typedef BlockSize<8,8> FastWaveUVBlockSize;

// stencil definition
fastWaveUUpdate.Init(
    "FastWaveU",
    dycoreRepository.calculationDomain(),
    StencilConfiguration<Real, FastWaveUVBlockSize>(),
    concatenate_stages(
        StencilStage<PPGradCorStage, KLoopTerrainCoordinates, IJBoundary<cIndented, 0,1, 0,1>>(),
        StencilStage<UStage, KLoopAtmosphere, IJBoundary<cComplete, 0,0, 0,0>>()
    ),
    create_context(
        Ref(dycoreRepository.u_out()),
        CRef(fastWaveRepository.u_pos()),
        CRef(dycoreRepository.uadvt()),
        CRef(dycoreRepository.rho_in()),
        CRef(fastWaveRepository.pp()),
        CRef(fastWaveRepository.fx()),
        CRef(fastWaveRepository.wgtfac()),
        CRef(fastWaveRepository.dzdx()),
        Scalar("dts", fastWaveRepository.dts()),
        Buffer<KRangeTerrainCoordinates, IJKRealField>("ppgradcor")
    )
);
```

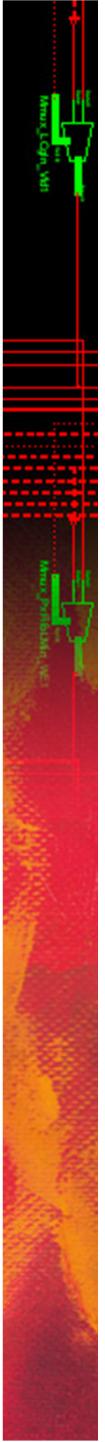
// stencil usage
for(int step=0; step<numberOfTimeSteps; ++step) fastWaveUUpdate.Apply();

3D loops are represented by stencil stages

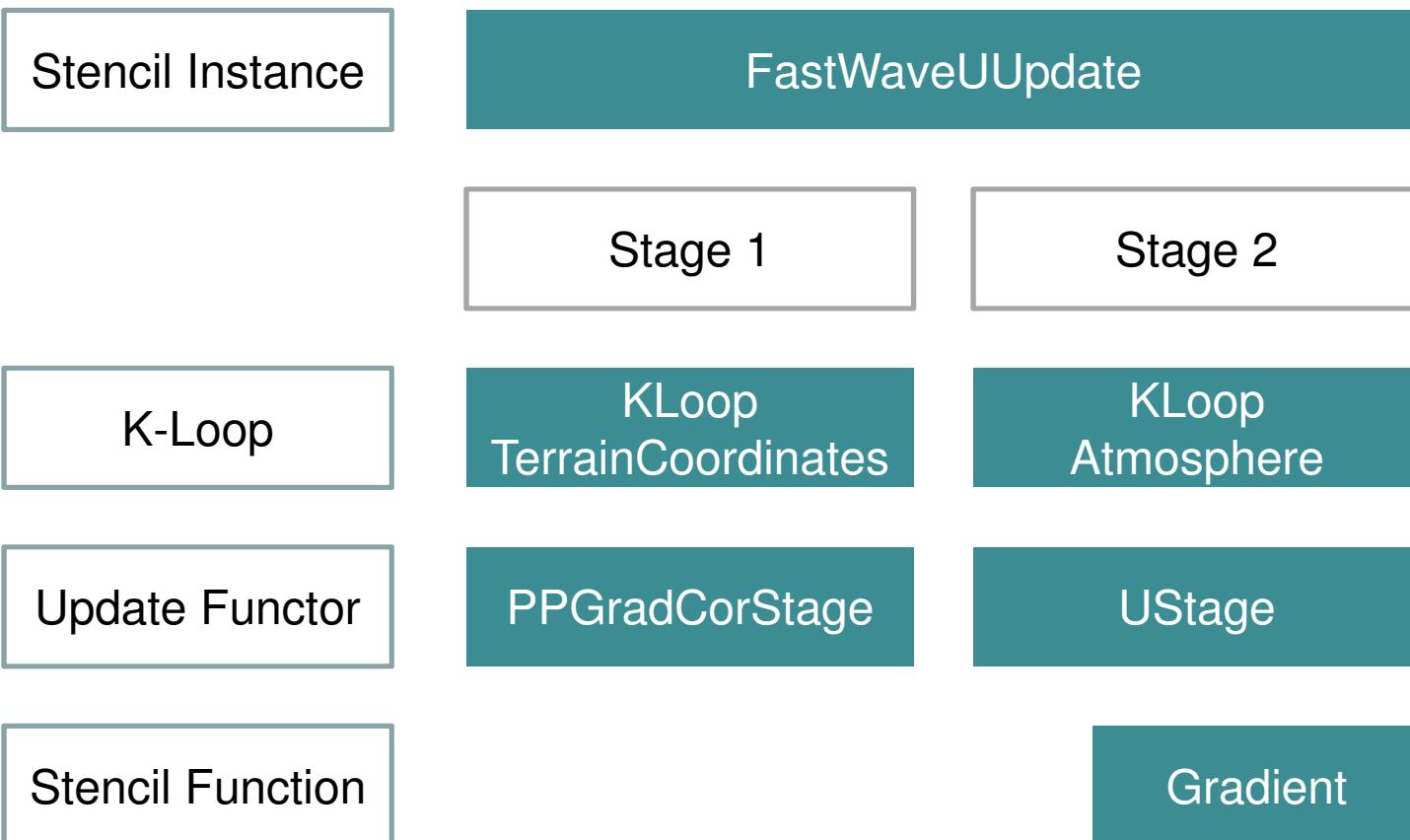
Update Functor K-Loop IJ - Boundary

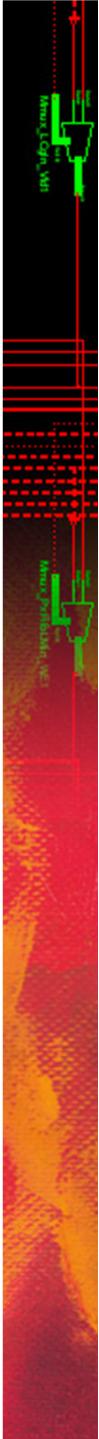
Define parameter tuple / context

Stencil stages can communicate efficiently using buffers

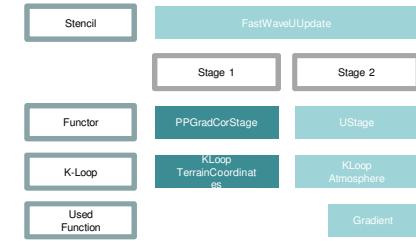


„Call Stack“





Stencil Stage Definition - Pressure Gradient Correction



```

template<typename TEnv>
struct PPGradCorStage
{
    STENCIL_STAGE(TEnv);

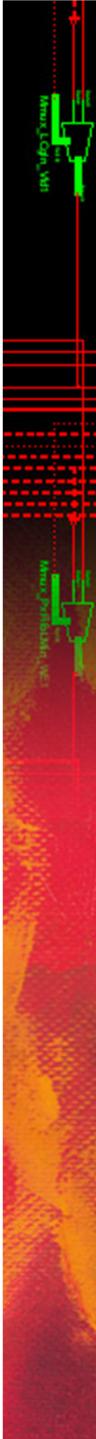
    STAGE_PARAMETER(TerrainCoordinates, 4, pp);
    STAGE_PARAMETER(TerrainCoordinates, 6, wgtfac); ← Parameter declaration
    STAGE_PARAMETER(TerrainCoordinates, 9, ppgradcor);

    USING(Delta2K);

    __ACC__ static void Do(Context ctx, TerrainCoordinates)
    {
        ctx[ppgradcor::Center()] = ctx[Delta2K::With(wgtfac::Center(), pp::Center())];
    }
};
  
```

Function computing
a weighted delta

Access fields via ctx object (setup using create tuple)



Stencil Stage Definition – U Update

```
template<typename TEnv>
struct UStage
{
    STENCIL_STAGE(TEnv);

    STAGE_PARAMETER(FullDomain, 0, u_out);
    STAGE_PARAMETER(FullDomain, 1, u_pos);
    STAGE_PARAMETER(FullDomain, 2, uadvt);
    STAGE_PARAMETER(FullDomain, 3, rho_in);
    STAGE_PARAMETER(FullDomain, 4, pp);
    STAGE_PARAMETER(FullDomain, 5, fx);
    STAGE_PARAMETER(FullDomain, 8, dts);
    STAGE_PARAMETER(TerrainCoordinates, 7, dzdx);
    STAGE_PARAMETER(TerrainCoordinates, 9, ppgradcor);

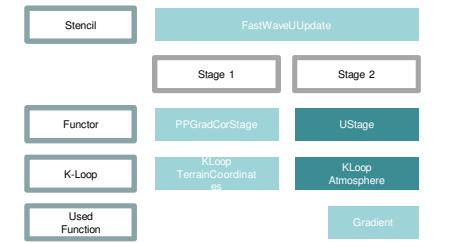
    USING(Average); USING(Delta); USING(Gradient);

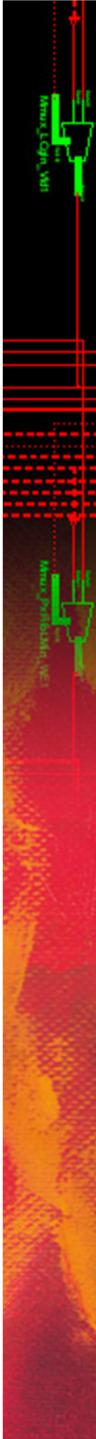
    __ACC__ static void Do(Context ctx, FullDomain)
    {
        T rho = ctx[fx::Center()] / ctx[Average::With(iplus1, rho_in::Center())];
        T pgrad = ctx[Gradient::With(iplus1,
            pp::Center(),
            Delta::With(kplus1, ppgradcor::Center()),
            dzdx::Center()
        )];

        ctx[u_out::Center()] = ctx[u_pos::Center()] +
            ctx[uadvt::Center()] - pgrad * rho * ctx[dts::Center()];
    }
};
```

Boulder, 8.9.2011, © by Supercomputing Systems

17





Stencil Function Definition – Gradient

```

template<typename TEnv>
struct Gradient
{
    STENCIL_FUNCTION(TEnv);

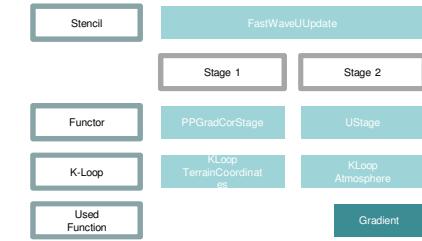
    FUNCTION_PARAMETER(0, dir);
    FUNCTION_PARAMETER(1, data);
    FUNCTION_PARAMETER(2, deltak);
    FUNCTION_PARAMETER(3, deltaz);

    USING(Delta); USING(Sum);

    __ACC__ static T Do(Context ctx, TerrainCoordinates)
    {
        return
            ctx[Delta::With(dir(), data::Center())] +
            ctx[Sum::With(dir(), deltak::Center())] * ctx[deltaz::Center()];
    }

    __ACC__ static T Do(Context ctx, FlatCoordinates)
    {
        // delta z is zero for the flat coordinates
        return
            ctx[Delta::With(dir(), data::Center())];
    }
};

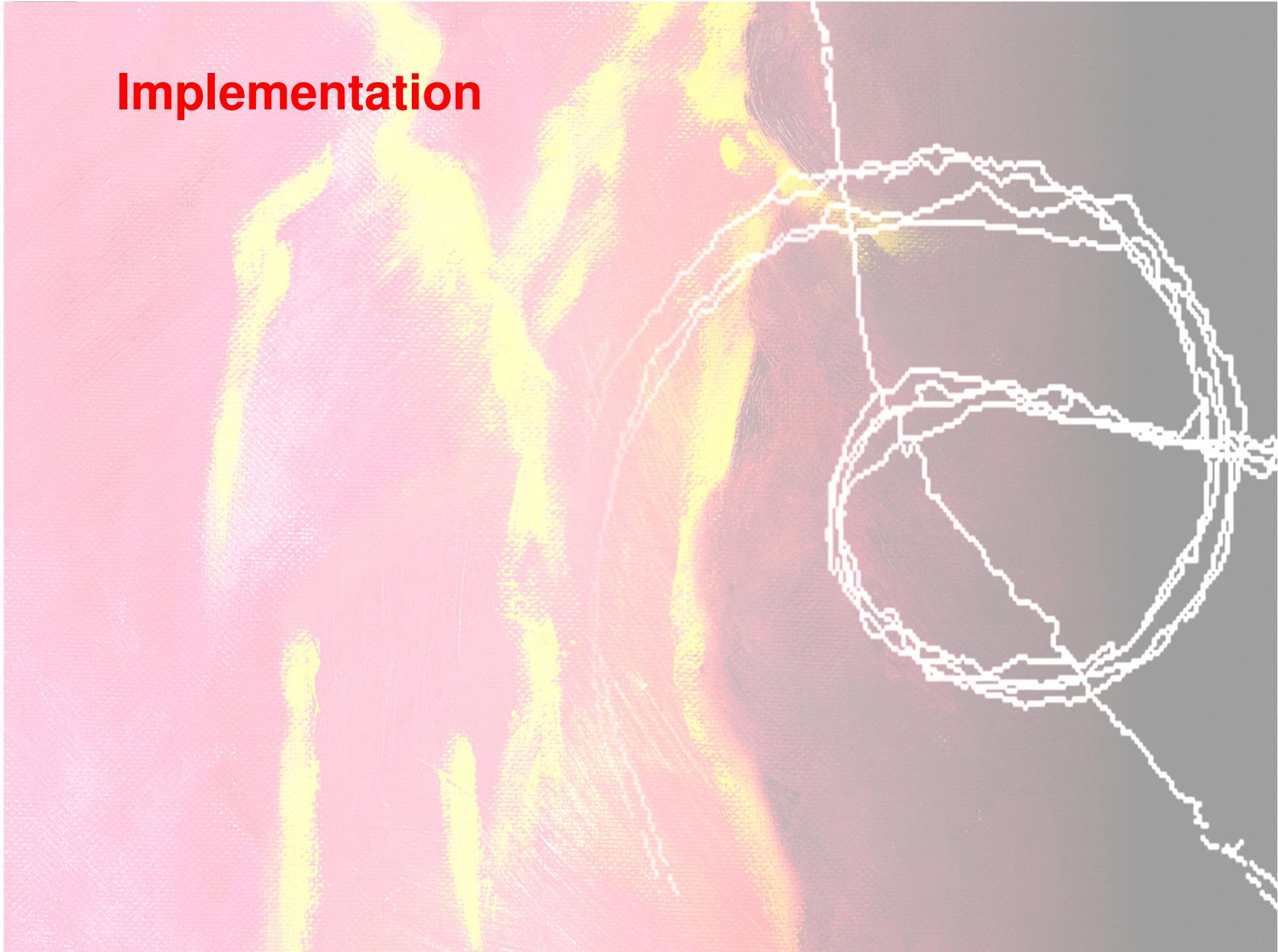
```

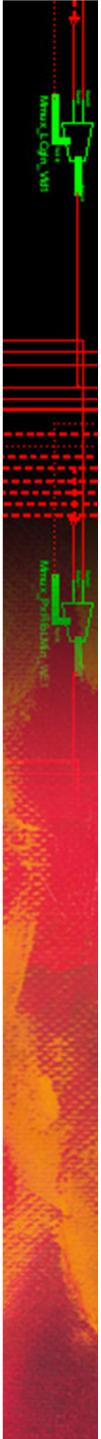


Stencil functions optionally return a value

Different update functions for different domains

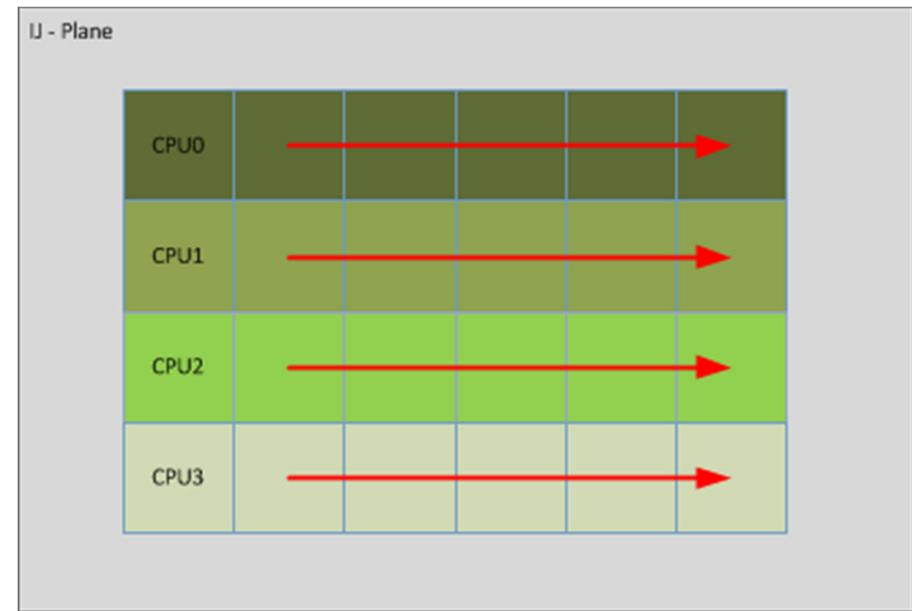
Implementation

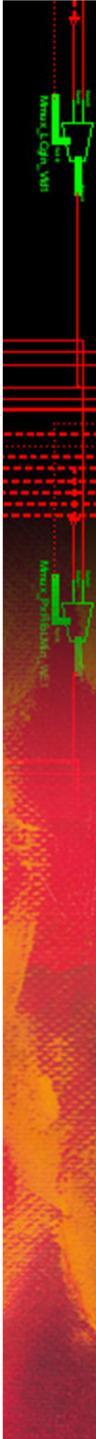




CPU Library Backend - Parallelization

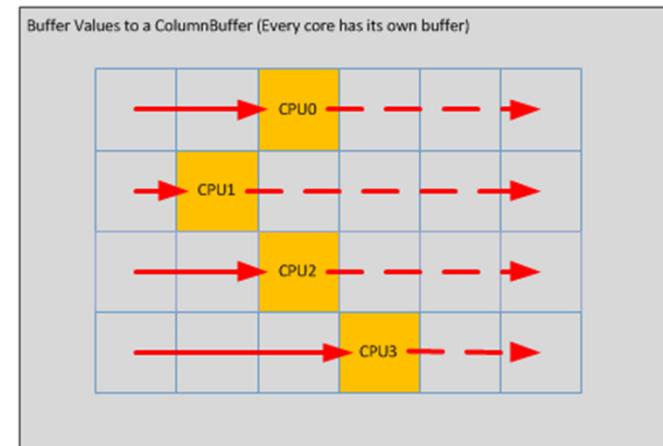
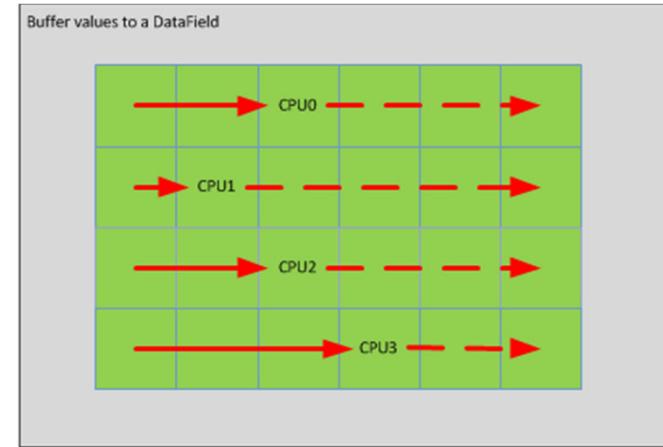
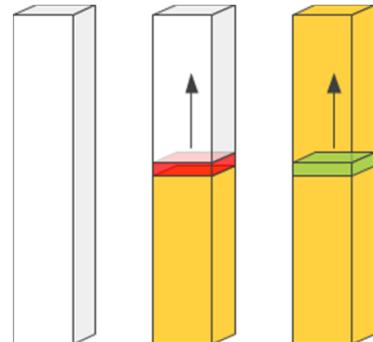
- Storage
 - The data is stored continuously in K
 - Works well for the numerous tri-diagonal solves in the vertical
 - Allows efficient blocking in the IJ-plane
- Loops
 - Loop over all blocks (parallelized with OpenMP)
 - Loop over all stencil stages
 - Loop over IJ
 - Loop over K



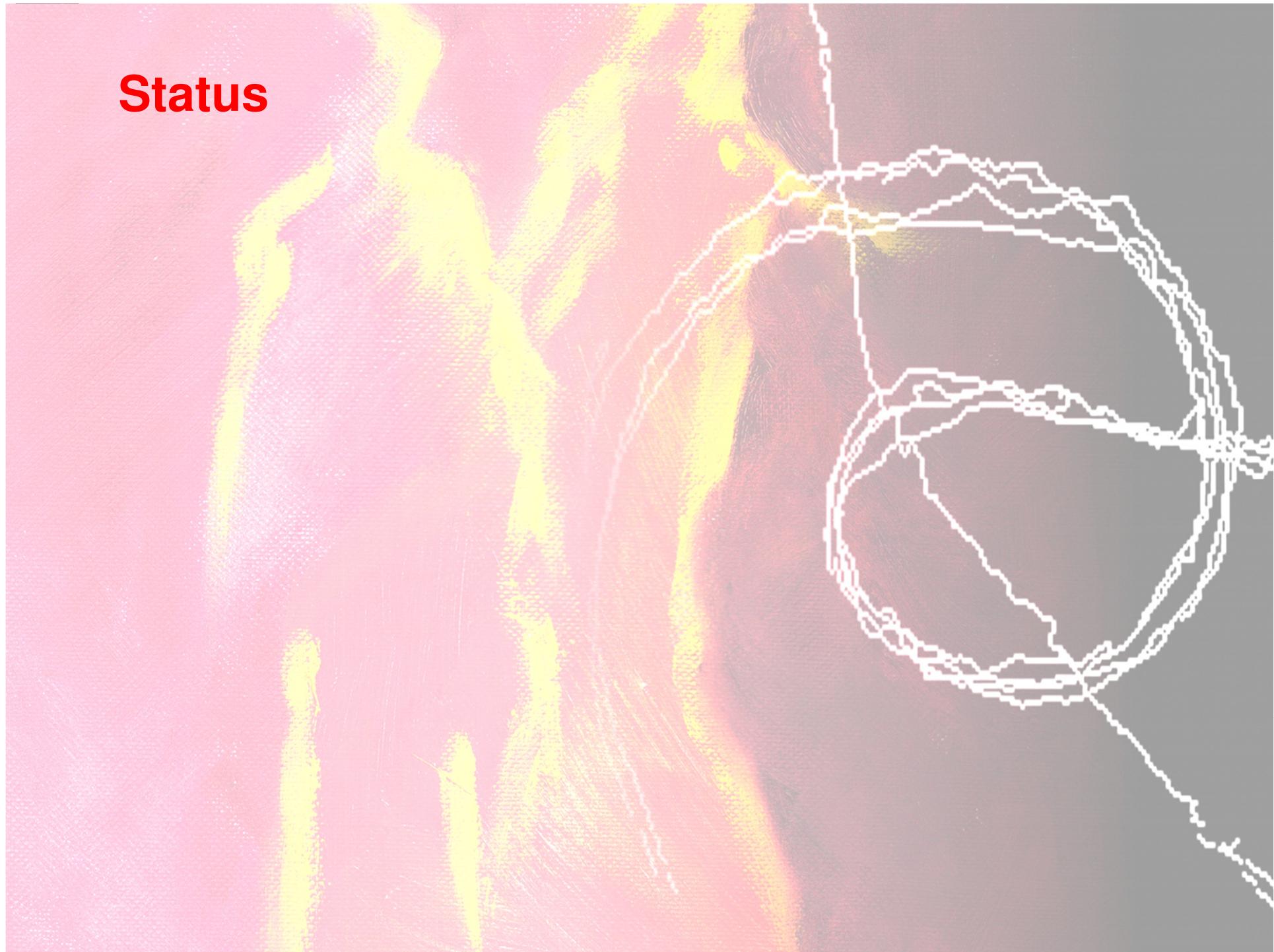


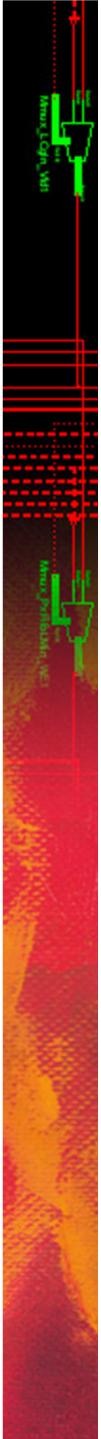
CPU Library Backend - Buffers

- Buffers store intermediate values handed over from one stage to the next
 - “ppgradcor” in the Fast Wave U-update
- COSMO uses full 3D fields
 - Intermediate value stored in DRAM
- Due to blocking it is possible to allocate a field with the size of a block for every core
 - Intermediate values are stored in L2 cache



Status

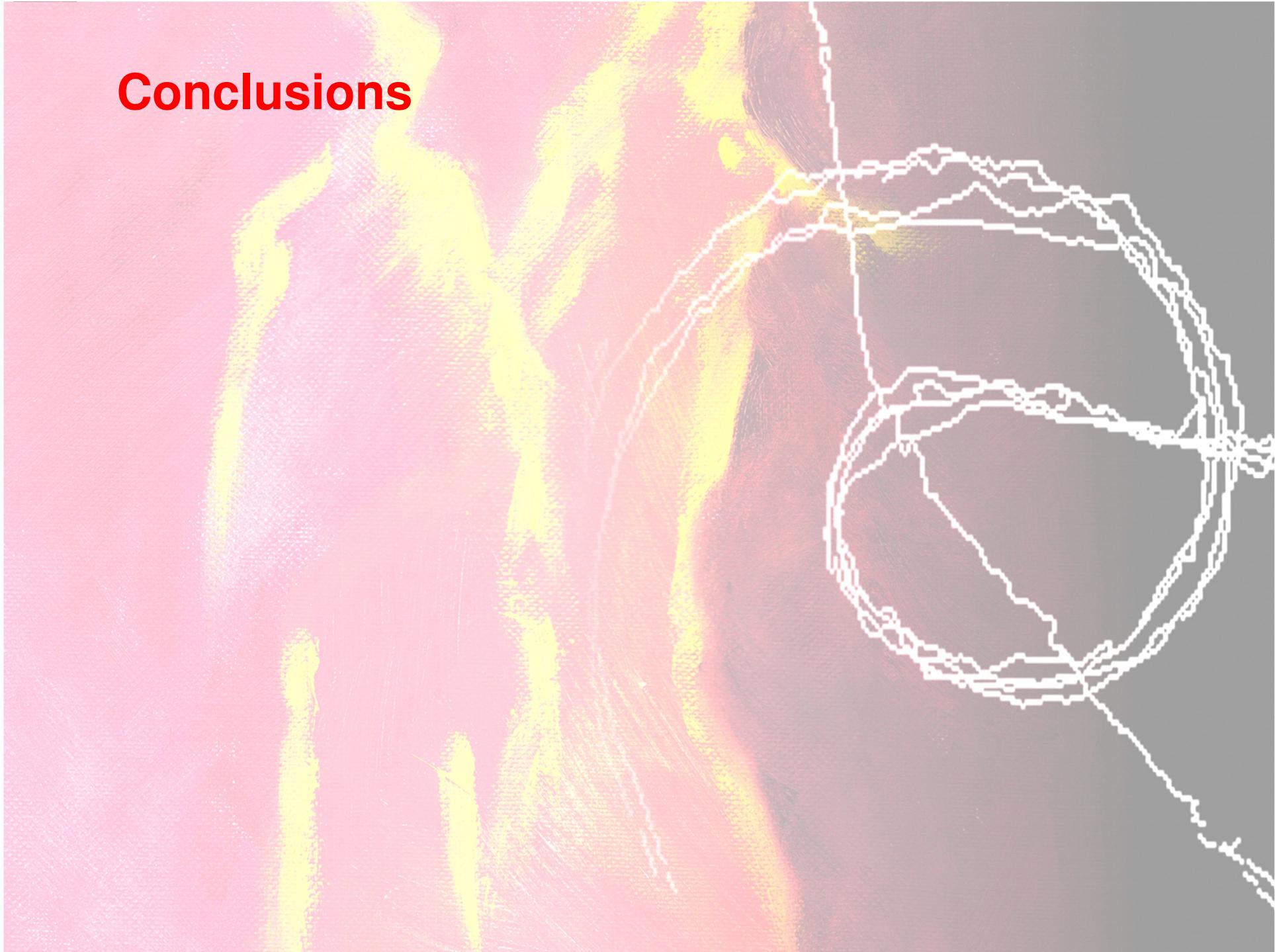


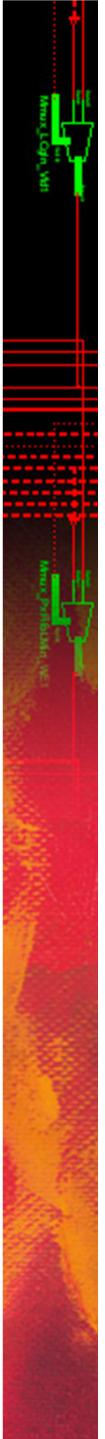


Status

- The HP2C dycore is functional and verified against COSMO
 - Fast Wave solver
 - Advection
 - 5th order advection
 - Bott 2 advection (cri implementation in z direction)
 - Implicit vertical diffusion and advection (slow tendencies)
 - Horizontal diffusion
 - Some smaller stencils like Coriolis, CalcRho etc.
- Performance portable CPU backend
 - Factor 1.5x – 1.6x faster than the standard COSMO implementation
 - SSE not used (expect another ~30%)
- Functional GPU / CUDA backend
 - No performance measurements so far
 - Room for performance tuning

Conclusions





Conclusions

- It works
 - Hardware platform specific code is hidden inside library
 - Performance portability feasible
 - Inside the library we could write “ugly” but fast code
 - Blocking
 - Iterators
 - The library has been successfully and efficiently used by experienced as well as ‘fresh’ domain scientists and SW engineers
- Work to be done
 - Achieve the expected GPU performance until end of 2011
 - Adoption of the code by the weather services / climate community

Discussion

Acknowledgements to all our collaborators at

- C2SM (Center for Climate Systems Modeling)
- CSCS
- DWD (Deutscher Wetterdienst)
- MeteoSwiss
- Nvidia